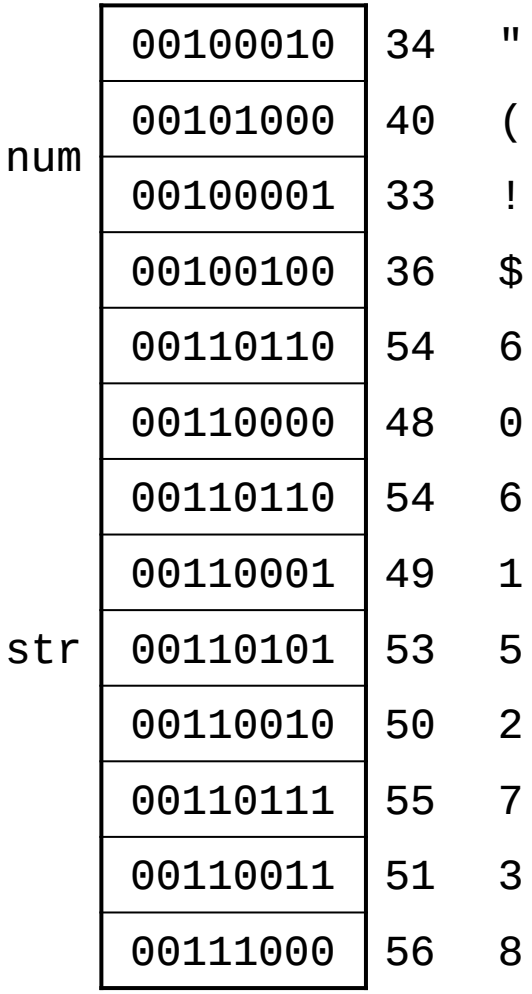
Binary Files

- `"606152738"` (`char *`) vs `606152738` (`int`)
  - `char *` takes 10 bytes (1 for each character + null)
  - `int` takes 4 bytes
    - 60 same size in bytes as 606152738
- << operator and `write()`
  - `outFile << number`
    - Outputs `number` (`int`) as a `char *`
  - `outFile.write( const char *, size );`
    - Copies data directly from memory into file

```
char str[] = "606152738";
int num = 606152738;
```

In memory



2 + 32 + 2048 + 8192 + 65536 + 2097152 + 67108864 + 536870912 = 606152738

$(00100100\ 00100001\ 00101000\ 00100010)_2 = (606152738)_{10}$

ASCII character set										
	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

```
char str[] = "606152738";
int num = 606152738;
```

In binary file

0	00100010	34	"
1	00101000	40	(
2	00100001	33	!
3	00100100	36	$
4	00110110	54	6
5	00110000	48	0
6	00110110	54	6
7	00110001	49	1
8	00110101	53	5
9	00110010	50	2
10	00110111	55	7
11	00110011	51	3
12	00111000	56	8

```
char str[] = "606152738";
int num = 606152738;
```

In text file

```
outFile << str << num;
```

0	00110110	54	6
1	00110000	48	0
2	00110110	54	6
3	00110001	49	1
4	00110101	53	5
5	00110010	50	2
6	00110111	55	7
7	00110011	51	3
8	00111000	56	8
9	00110110	54	6
10	00110000	48	0
11	00110110	54	6
12	00110001	49	1
13	00110101	53	5
14	00110010	50	2
15	00110111	55	7
16	00110011	51	3
17	00111000	56	8

Creating a Binary File

- Example

```
outFile.write( reinterpret_cast< const char * >( &number ),
               sizeof( number ) );
```

  - `&number` is an `int *`
    - Convert to `const char *` with `reinterpret_cast`
  - `sizeof(number)`
    - Size of `number` (an `int`) in bytes
  - `read` function similar (more later)
  - Use `ios::binary`

```
struct StudentData
{
   char studentId[ 8 ];
   char lastName[ 12 ];
   char firstName[ 12 ];
   int grade;
};

int main()
{
   ofstream outGrade( "grades.dat", ios::out | ios::binary );
   if( !outGrade )
   {
      cout << "File could not be opened." << endl;
      exit( 1 );
   }
   StudentData students[ 5 ] = { { "1121411", "Tseng",  "Dora", 91 },
                                { "1121429",    "Hu",   "Jay", 70 },
                                { "1121430",  "Wong", "Ariel", 91 },
                                { "1121432", "Chang", "Eason", 83 },
                                { "1121443",   "Lin",   "jet", 70 } };
   for( int i = 0; i < 5; ++i )
      outGrade.write( reinterpret_cast< const char * >( &students[ i ] ),
                      sizeof( StudentData ) );
}
```

Creating a Binary File

- Usually write entire `struct` or object to file
- Problem statement
  - Credit processing program
  - Store at most 100 fixed-length records
  - Record
    - Account number (key)
    - First and last name
    - Balance
  - Account operations
    - Update, create new, delete, list all accounts in a file
- Next: program to create blank 100-record file

```
struct StudentData
{
   char studentId[ 8 ];
   char lastName[ 12 ];
   char firstName[ 12 ];
   int grade;
};

int main()
{
   ofstream outGrade( "grades.dat", ios::out | ios::binary );

   if( !outGrade )
   {
      cerr << "File could not be opened." << endl;
      system( "pause" );
      exit( 1 );
   }

   StudentData blankStudent = { "", "", "", 0 };

   for( int i = 0; i < 50; i++ )
      outGrade.write( reinterpret_cast< const char * >( &blankStudent ),
                      sizeof( StudentData ) );
}
```

```
int main()
{
   ifstream inGrade( "grades.dat", ios::in | ios::binary );
   if( !inGrade )
   {
      cout << "File could not be opened." << endl;
      exit( 1 );
   }
   StudentData student;
   inGrade.read( reinterpret_cast< char * >( &student ),
                 sizeof( StudentData ) );
   while( !inGrade.eof() )
   {
      cout << left << setw( 13 ) << student.studentId
           << setw( 12 ) << student.lastName
           << setw( 13 ) << student.firstName
           << setw( 5 ) << right << student.grade << endl;

      inGrade.read( reinterpret_cast< char * >( &student ),
                    sizeof( StudentData ) );
   }
   inGrade.close();
}
```

```
int main()
{
   ifstream inGrade( "grades.dat", ios::in | ios::binary );

   if( !inGrade )
   {
      cout << "File could not be opened." << endl;
      exit( 1 );
   }

   StudentData student;

   while( inGrade.read( reinterpret_cast< char * >( &student ),
          sizeof( StudentData ) ) )
   {
      cout << left << setw( 13 ) << student.studentId
           << setw( 12 ) << student.lastName
           << setw( 13 ) << student.firstName
           << setw( 5 ) << right << student.grade << endl;
   }

   inGrade.close();
}
```

Writing Data Randomly to a Binary File

- Use `seekp` to write to exact location in file
  - Where does the first record begin?
    - Byte 0
  - The second record?
    - `sizeof( object )`
  - Any record?
    - `( Recordnum - 1 ) * sizeof( object )`

```
struct StudentData
{
   char studentId[ 8 ];
   char lastName[ 12 ];
   char firstName[ 12 ];
   int grade;
};

int main()
{
   fstream outGrade( "grades.dat", ios::in | ios::out | ios::binary );

   if( !outGrade )
   {
      cerr << "File could not be opened." << endl;
      system( "pause" );
      exit( 1 );
   }

   cout << "Enter student ID ( 0 to end input )\n? ";

   StudentData student;
```

```
   cin >> student.studentId;

   while( strcmp( student.studentId, "1111401" ) >= 0 &&
          strcmp( student.studentId, "1111450" ) <= 0 )
   {
      cout << "Enter lastname, firstname, grade\n? ";
      cin >> student.lastName;
      cin >> student.firstName;
      cin >> student.grade;

      int inClassId = 10 * ( student.studentId[ 5 ] - '0' )
                         + ( student.studentId[ 6 ] - '0' );

      outGrade.seekp( ( inClassId - 1 ) * sizeof( StudentData ) );

      outGrade.write( reinterpret_cast< const char * >( &student ),
                      sizeof( StudentData ) );

      cout << "Enter student ID\n? ";
      cin >> student.studentId;
   }
}
```

```
Enter student ID ( 0 to end input )
? 1111401
Enter lastname, firstname, grade
? Smith James 52
Enter student ID
? 1111402
Enter lastname, firstname, grade
? Jones Robert 42
Enter student ID
? 1111403
Enter lastname, firstname, grade
? Taylor John 60
Enter student ID
? 1111404
Enter lastname, firstname, grade
? Brown Michael 67
Enter student ID
? 1111405
Enter lastname, firstname, grade
? Williams David 76
Enter student ID
? 0
```

Reading from a Binary File Sequentially

- `read` - similar to `write`
  - `inFile.read( reinterpret_cast< char * >( &number ), sizeof( int ) );`
    - `&number`: location to store data
    - `sizeof(int)`: how many bytes to read

```
struct StudentData
{
   char studentId[ 8 ];
   char lastName[ 12 ];
   char firstName[ 12 ];
   int grade;
};

int main()
{
   ifstream inGrade( "grades.dat", ios::in | ios::binary );

   if( !inGrade )
   {
      cerr << "File could not be opened." << endl;
      system( "pause" );
      exit( 1 );
   }
```

```
   cout << setw( 13 ) << left << "student ID"
        << setw( 12 ) << "Last Name"
        << setw( 13 ) << "First Name"
        << setw( 5 ) << right << "Grade" << endl;

   StudentData student;
   inGrade.read( reinterpret_cast< char * >( &student ),
                 sizeof( StudentData ) );

   while( !inGrade.eof() )
   {
      cout << setw( 13 ) << left << student.studentId
           << setw( 12 ) << student.lastName
           << setw( 13 ) << student.firstName
           << setw( 5 ) << right << student.grade << endl;

      inGrade.read( reinterpret_cast< char * >( &student ),
                    sizeof( StudentData ) );
   }
}
```

student ID	Last Name	First Name	Grade
1111401	Smith	James	52
1111402	Jones	Robert	42
1111403	Taylor	John	60
1111404	Brown	Michael	67
1111405	Williams	David	76

Case Study: A Transaction-Processing Program

- Give user menu
  - Option 1: store accounts to **`print.txt`**

```
Account   Last Name       First Name    Balance
29        Brown           Nancy          -24.54
33        Dunn            Stacey         314.33
37        Barker          Doug             0.00
88        Smith           Dave           258.34
96        Stone           Sam             34.98
```

  - Option 2: update record

```
Enter account to update (1 - 100): 37
37        Barker          Doug             0.00

Enter charge (+) or payment (-): +87.99
37        Barker          Doug            87.99
```

Case Study: A Transaction-Processing Program

- Menu options (continued)
  - Option 3: add new record

```
Enter new account number (1 - 100): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```

  - Option 4: delete record

```
Enter account to delete (1 - 100): 29
Account #29 deleted.
```

```
1   // Fig. 17.7: fig17_07.cpp
2   // This program reads a random access file sequentially, updates
3   // data previously written to the file, creates data to be placed
4   // in the file, and deletes data previously in the file.
5   #include <iostream>
6   #include <iomanip>
7   #include <fstream>
8   Using namespace std;
9
10  struct ClientData
11  {
12     int accountNumber;
13     char lastName[ 15 ];
14     char firstName[ 10 ];
15     double balance;
16  }; // end struct ClientData
17
18  int enterChoice();
19  void creatTextFile( fstream& );
20  void updateRecord( fstream& );
21  void newRecord( fstream& );
```

```
22    void deleteRecord( fstream& );
23    void outputLine( const ClientData );
24    int getAccount( const char * const );
25
26    enum Choices { PRINT = 1, UPDATE, NEW, DELETE, END };
27
28    int main()
29    {
30       // open file for reading and writing
31       fstream inOutCredit( "credit.dat", ios::in | ios::out |
                                            ios::binary );
32
33       // exit program if fstream cannot open file
34       if ( !inOutCredit )
35       {
36          cerr << "File could not be opened." << endl;
37          exit ( 1 );
38       } // end if
39
40       int choice; // store user choice
41
```

```
42       // enable user to specify action
43       while ( ( choice = enterChoice() ) != END )
44       {
45          switch ( choice )
46          {
47             case PRINT: // create text file from record file
48                creatTextFile( inOutCredit );
49                break;
50             case UPDATE: // update record
51                updateRecord( inOutCredit );
52                break;
53             case NEW: // create record
54                newRecord( inOutCredit );
55                break;
56             case DELETE: // delete existing record
57                deleteRecord( inOutCredit );
58                break;
59             default: // display error if user does not select valid choice
60                cerr << "Incorrect choice" << endl;
61                break;
62          } // end switch
63
64          inOutCredit.clear(); // reset end-of-file indicator
65       } // end while
66    } // end main
```

```
67
68    // enable user to input menu choice
69    int enterChoice()
70    {
71       // display available options
72       cout << "\nEnter your choice" << endl
73            << "1 - store a formatted text file of accounts" << endl
74            << "    called \"print.txt\" for printing" << endl
75            << "2 - update an account" << endl
76            << "3 - add a new account" << endl
77            << "4 - delete an account" << endl
78            << "5 - end program\n? ";
79
80       int menuChoice;
81       cin >> menuChoice; // input menu selection from user
82       return menuChoice;
83    } // end function enterChoice
84
```

```
85  // create formatted text file for printing
86  void creatTextFile( fstream &readFromFile )
87  {
88     // create text file
89     ofstream outPrintFile( "print.txt", ios::out );
90
91     // exit program if ofstream cannot create file
92     if ( !outPrintFile )
93     {
94        cerr << "File could not be created." << endl;
95        exit( 1 );
96     } // end if
97
98     outPrintFile << left << setw( 10 ) << "Account" << setw( 16 )
99         << "Last Name" << setw( 11 ) << "First Name" << right
100        << setw( 10 ) << "Balance" << endl;
101
102     // set file-position pointer to beginning of readFromFile
103     readFromFile.seekg( 0 );
104
```

```
105     // read first record from record file
106     ClientData client;
107     readFromFile.read( reinterpret_cast< char * >( &client ),
108                        sizeof( ClientData ) );
109
110     // copy all records from record file into text file
111     while ( !readFromFile.eof() )
112     {
113        // write single record to text file
114        if ( client.accountNumber != 0 ) // skip empty records
115           outPrintFile << left << setw( 10 )
116                        << client.accountNumber << setw( 16 )
117                        << client.lastName << setw( 11 )
118                        << client.firstName << setw( 10 )
119                        << setprecision( 2 ) << right << fixed
120                        << showpoint << client.balance << endl;
121
122        // read next record from record file
123        readFromFile.read( reinterpret_cast< char * >( &client ),
124                           sizeof( ClientData ) );
125     } // end while
126  } // end function creatTextFile
127
```

```
128 // update balance in record
129 void updateRecord( fstream &updateFile )
130 {
131    // obtain number of account to update
132    int accountNumber = getAccount( "Enter account to update" );
133
134    // move file-position pointer to correct record in file
135    updateFile.seekg(
          ( accountNumber - 1 ) * sizeof( ClientData ) );
136
137    // read first record from file
138    ClientData client;
139    updateFile.read( reinterpret_cast< char * >( &client ),
140       sizeof( ClientData ) );
141
142    // update record
143    if ( client.accountNumber != 0 )
144    {
145       outputLine( client ); // display the record
146
```

```
147         // request user to specify transaction
148         cout << "\nEnter charge (+) or payment (-): ";
149         double transaction; // charge or payment
150         cin >> transaction;
151
152         // update record balance
153         double oldBalance = client.balance;
154         client.balance = oldBalance + transaction;
155         outputLine( client ); // display the record
156
157         // move file-position pointer to correct record in file
158         updateFile.seekp(
               ( accountNumber - 1 ) * sizeof( ClientData ) );
159
160         // write updated record over old record in file
161         updateFile.write(
               reinterpret_cast< const char * >( &client ),
162            sizeof( ClientData ) );
163      } // end if
164      else // display error if account does not exist
165         cerr << "Account #" << accountNumber
166              << " has no information." << endl;
167   } // end function updateRecord
168
```

```
169 // create and insert record
170 void newRecord( fstream &insertInFile )
171 {
172    // obtain number of account to create
173    int accountNumber = getAccount( "Enter new account number" );
174
175    // move file-position pointer to correct record in file
176    insertInFile.seekg(
          ( accountNumber - 1 ) * sizeof( ClientData ) );
177
178    // read record from file
179    ClientData client;
180    insertInFile.read( reinterpret_cast< char * >( &client ),
181       sizeof( ClientData ) );
182
183    // create record, if record does not previously exist
184    if ( client.accountNumber == 0 )
185    {
186       string lastName;
187       string firstName;
188       double balance;
189
```

```
190         // user enters last name, first name and balance
191         cout << "Enter lastname, firstname, balance\n? ";
192         cin >> setw( 15 ) >> lastName;
193         cin >> setw( 10 ) >> firstName;
194         cin >> balance;
195
196         // use values to populate account values
197         lastName.copy( client.lastName, lastName.size() );
198         firstName.copy( client.firstName, firstName.size() );
199         client.balance = balance;
200         client.accountNumber = accountNumber;
201
202         // move file-position pointer to correct record in file
203         insertInFile.seekp( ( accountNumber - 1 ) *
               sizeof( ClientData ) );
204
205         // insert record in file
206         insertInFile.write(
               reinterpret_cast< const char * >( &client ),
207            sizeof( ClientData ) );
208      } // end if
209      else // display error if account previously exists
210         cerr << "Account #" << accountNumber
211              << " already contains information." << endl;
212   } // end function newRecord
```

```
183     // create record, if record does not previously exist
184     if ( client.accountNumber == 0 )
185     {
190        // user enters last name, first name and balance
191        cout << "Enter lastname, firstname, balance\n? ";
192        cin >> setw( 15 ) >> client.lastName;
193        cin >> setw( 10 ) >> client.firstName;
194        cin >> client.balance;
200        client.accountNumber = accountNumber;
201
202        // move file-position pointer to correct record in file
203        insertInFile.seekp( ( accountNumber - 1 ) *
              sizeof( ClientData ) );
204
205        // insert record in file
206        insertInFile.write(
              reinterpret_cast< const char * >( &client ),
207           sizeof( ClientData ) );
208     } // end if
209     else // display error if account previously exists
210        cerr << "Account #" << accountNumber
211             << " already contains information." << endl;
212  } // end function newRecord
```

```
213
214 // delete an existing record
215 void deleteRecord( fstream &deleteFromFile )
216 {
217    // obtain number of account to delete
218    int accountNumber = getAccount( "Enter account to delete" );
219
220    // move file-position pointer to correct record in file
221    deleteFromFile.seekg(
          ( accountNumber - 1 ) * sizeof( ClientData ) );
222
223    // read record from file
224    ClientData client;
225    deleteFromFile.read( reinterpret_cast< char * >( &client ),
226                         sizeof( ClientData ) );
227
```

```
228     // delete record, if record exists in file
229     if ( client.accountNumber != 0 )
230     {  // create blank record
231        ClientData blankClient = { 0, "", "", 0.0 };
232
233        // move file-position pointer to correct record in file
234        deleteFromFile.seekp( ( accountNumber - 1 ) *
235           sizeof( ClientData ) );
236
237        // replace existing record with blank record
238        deleteFromFile.write(
239           reinterpret_cast< const char * >( &blankClient ),
240           sizeof( ClientData ) );
241
242        cout << "Account #" << accountNumber << " deleted.\n";
243     } // end if
244     else // display error if record does not exist
245        cerr << "Account #" << accountNumber << " is empty.\n";
246  } // end deleteRecord
247
```

```
248 // display single record
249 void outputLine( const ClientData record )
250 {
251    cout << left << setw( 10 ) << record.accountNumber
252         << setw( 16 ) << record.lastName
253         << setw( 11 ) << record.firstName
254         << setw( 10 ) << setprecision( 2 ) << right << fixed
255         << showpoint << record.balance << endl;
256 } // end function outputLine
257
258 // obtain account-number value from user
259 int getAccount( const char * const prompt )
260 {
261    int accountNumber;
262
263    // obtain account-number value
264    do
265    {
266       cout << prompt << " (1 - 100): ";
267       cin >> accountNumber;
268    } while ( accountNumber < 1 || accountNumber > 100 );
269
270    return accountNumber;
271 } // end function getAccount
```

A Tip

```cpp
// Fig. 17.5: fig17_05.cpp
// Writing to a random access file.
#include <iostream>
#include <iomanip>
#include <fstream>
Using namespace std;

struct ClientData
{
   int accountNumber;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;
}; // end struct ClientData

```

```
16 int main()
17 {
18    ClientData client;
19    cout << "Enter account number "
20         << "(1 to 100, 0 to end input)\n? ";
21
22    cin >> client.accountNumber;
23
24    fstream inoutCredit( "credit.dat", ios::in | ios::out |
25                                       ios::binary );
26
27    if ( !inoutCredit )
28    {
29       ofstream outCredit( "credit.dat", ios::binary );
30
31       if ( !outCredit )
32       {
33          cerr << "File could not be opened." << endl;
34          exit( 1 );
35       } // end if
```

```
36
37          while ( accountNumber > 0 && accountNumber <= 100 )
38          {
39             cout << "Enter lastname, firstname, balance\n? ";
40             cin >> setw( 15 ) >> client.lastName;
41             cin >> setw( 10 ) >> client.firstName;
42             cin >> client.balance;
43
44             outCredit.seekp( ( client.accountNumber - 1 ) *
45                sizeof( ClientData ) );
46
47             outCredit.write(
48                reinterpret_cast< const char * >( &client ),
49                sizeof( ClientData ) );
50
51             cout << "Enter account number\n? ";
52             cin >> accountNumber;
53          } // end while
54       } // end if
```

```
55     else
56     {
57        while ( accountNumber > 0 && accountNumber <= 100 )
58        {
59           cout << "Enter lastname, firstname, balance\n? ";
60           cin >> setw( 15 ) >> client.lastName;
61           cin >> setw( 10 ) >> client.firstName;
62           cin >> client.balance;
63
64           inoutCredit.seekp( ( client.accountNumber - 1 ) *
65              sizeof( ClientData ) );
66
67           inoutCredit.write(
68              reinterpret_cast< const char * >( &client ),
69              sizeof( ClientData ) );
70
71           cout << "Enter account number\n? ";
72           cin >> accountNumber;
73        } // end while
74     } // end else
75  } // end main
```